General and efficient decentralized consensus protocols

Jean-Claude Bermond, Jean-Claude König, Michel Raynal

General and Efficient Decentralized Consensus Protocols

Bermond Jean-Claude and Konig Jean-Claude
LRI, Informatique, U.A. C.N.R.S. 410, bât 490,
Université Paris-Sud,91405 Orsay (France)

Raynal Michel
IRISA, Université de Rennes,
Campus de Beaulieu, 35042 Rennes (France)

Abstract :

In this article we are interested in computing a function or a predicate whose arguments are distributed on the nodes (or processors) of a network. When the computation is finished two cases may occur according to the application. The result is either known by each node or distributed on each node. Furthermore during the computation all the nodes have the same behaviour (there is no privileged node). We shall call the distributed algorithms which achieve such computations "consensus protocols".

A general and efficient consensus protocol is presented here. It is based on the concepts of phases and filterings. This algorithm can be applied to many problems, for example to compute a minimum routing table for the nodes of the network or to find the maximum of the identities of the nodes (election). If we denote by D the diameter of the network and by m the number of channels (communication links), its message complexity is at most $2(D+1)m$ and its time complexity is at most $(2D+1)\tau$, where τ is the maximum transmission delay on a channel.

This work has been done with the support of the french GRECO-PRC C^3.

1. Consensus protocols

1.1.Transformation problems

It is now usual to consider two kinds of systems according to their functionality and behaviour [HP85]. The first class consists of the systems called "reactive systems". Indeed they react to external sollicitations in such a way that some relations are always satisfied (invariants properties). For example the operating systems belong to this class. The second class consists of the systems called "transformation systems". They take data as input and make some transformations on the data in order to obtain a result. In particular, such systems implement the computation of a function. Here we consider the second class in the context of distributed applications and networks [R87]. Such a network will be modelled as a graph G=(V,E) with the vertex set V representing the nodes (or processors) and the edge set E representing the communication links or channels.

1.2. General hypotheses

The network consists of n nodes. We have no hypothesis on its topology except that it is connected and that its topology does not change during the execution of the algorithm. The channels are supposed to be bidirectionnal. That is the associated graph is undirected. The number of channels will be denoted by m, the distance between the nodes x and y by d(x,y) and the diameter (which is the maximum of the distance over any pair of nodes) by D.

The network is supposed to be asynchronous: there is no central controller like shared memory or global clock. Nodes communicate only by passing messages. Messages are transmitted without errors or duplications. They are received after a finite time, in the order in which they were sent (however we will see that this hypothesis is not necessary). Processing time is negligible compared to transmission delays.

Initially each node knows only its own identity, the channels incident with itself and furthermore it can distinguish among them. No other topological information is known by the nodes. For example no node knows the total number of nodes. Such a hypothesis will insure that the protocol is "modular" [B87]. A modification of the network will not impose to redefine all the network and the associated protocols.

1.3. The consensus protocols

In such a protocol the initial data are distributed on the nodes of the networks. In order to compute the result, the nodes must cooperate and coordinate their activities. Furthermore we want the cooperation control to be decentralized. In other words, no node (or subset of nodes) can have a privileged behaviour. The protocol in a given node depends only on the initial knowledge of the node. Therefore the algorithm is the same on each node (symmetry properties [B87]). When the distributed computation is terminated the result is either known by each node or distributed on each node.The situation depends on the application. If the aim is to compute a function whose result R is unique, then each node must obtain a copy of R. For example R can be the diameter of the network, the maximum value of the data (election of a leader) or the sum of the data. Otherwise if the (global) result R is the union of (partial) results r_i which are only useful to node P_i, then all that the node P_i must know is r_i. It happens for example when one wants to compute the eccentricity of a node (the eccentricity of a node is the maximum of the distances between it and all other nodes) or the table of minimum routings (each node only has to know on which channel it must route the messages to the others nodes).

We will call such distributed algorithms **"consensus protocols"** . The word "consensus" indicates both the node agreement on the result and the rules of cooperation.

The aim of the article is to give algorithmic principles of a general and efficient consensus protocol.

2. Fundamental algorithmic concepts

2.1. Notion of phase

Recall that the nodes must have the same behaviour (symmetry) and only a local knowledge of their incident channels (modularity). Therefore the algorithm will use "exchange" phases to insure cooperation between the nodes. For a given node a phase consists of three parts :

- emission of information to its neighbors (via incident channels)
- reception of information from its neighbors (via incident channels)
- local computation, memorization, and so on.

The algorithm is split into three classical parts:

- initialization (INIT)
- several phases (PHASES)*
- termination (TERM)

Let us note that the use of phases induces a logical synchronization ([G82] and [A85]). We can suppose that all the nodes start the algorithm simultaneously (in a logical time). If it is not the case, then we have to use a classical wake up procedure. When a node receives its first message, then it knows that the algorithm has begun. Therefore it executes its initialization part and begins the first phase. At the end of the first phase each node has received a message from all its neighbors. We will see in the proof that at the end of phase p each node has received the information contained in the nodes at distance at most p from itself.

The termination problem is more complicated. If the nodes know the diameter D of the network (in that case they have some global information), then after D phases the algorithm terminates. Otherwise if a node only has local knowledge, it must iterate the phase part till it no longer receives new information. Thus the nodes do not necessarily terminate at the end of the same phase: it depends on their eccentricity. Furthermore a node has to inform its neighbors that it has finished. It can do that for example by sending a type "end" message. We will see later that two nodes can use the received information to close the channel between them if they have no more information to learn through this channel.

2.2 Filtering notions

In view of the structure of the phases a node needs only to send at phase p the new information that he received during the phase $p-1$. Furthermore it does not have to transmit on a channel the new information that it received on this channel. So in this first filtering we eliminate redundant information. In fact we can also use a more clever filtering to close channels.

Let us call *sent(c)* (resp. *received(c)*) the information sent (resp. received) by a node P at phase p on channel c.

If *sent(c)* is equal to *received(c)* that means that the nodes P and Q joined by c know the same information at the end of the preceding phase and we will see in the proof that they will never learn any new information through channel c and therefore we can close channel c.

Otherwise P learns the information contained in *received(c) -sent(c)* through channel c at phase p. If this information happens to be included in the information that P already knows, then P will send ø at the next phase through that channel. We will see in remark 3.7 that we can avoid to send (and receive) more that one empty message by closing the channel in only one direction. However we need in that case to distinguish between emission and reception channels.

P stops the execution of the algorithm when all the channels are closed.

Note that the two filterings correspond to the two important concepts of distributed systems, namely the messages and the channels.

Our algorithm can easily be adapted to the case of directed networks, but in that case we cannot use the filterings.

2.3. One example

We will give here an example which illustrates the notions of phase and filtering of type 1. To avoid complications in writing the algorithm we will suppose that each node knows the value D of the diameter (in that case the termination problem is easy). This hypothesis will be relaxed in the general algorithm.

Suppose that we want to obtain the optimal routing table. At the end of the algorithm each node must know which channel it has to use in order to route a message to each other node. That can be done for example by associating with channel c the subset of nodes for which the first edge of the route is c.

The algorithm is given for an arbitrary node P.

p : number of the current phase.

c : any channel incident to P.

Inf : The global information known by P (it is a set consisting of the identities of the nodes for which P knows a shortest route).

New : The new information obtained by P since the beginning of the current phase.

sent(c) : message sent on channel *c* at the current phase.

received(c) : message received through channel *c* at the current phase.

```
INIT      p ← 0
          Inf ← {identity of the node}
          sent(c) ← Inf  for every channel c

PHASES    While p <D do
          begin
            p ← p +1
            send <sent(c) > on every channel c
            New ← ø
            For every channel c do
            begin
                receive < received(c) > on c
                for every y in received(c) - Inf - New : Rout(c) ← y ∪Rout(c)
                New ← New ∪ (received(c) - Inf )
            end
            Inf ← Inf ∪New
            sent(c)  ← New - received(c)
          end

TERM      Rout defines the table of minimum routing in the considered node.
          Inf gives the set of identities of all nodes and its cardinality the number of
          nodes.
```

3. A general Algorithm

3.1 The Algorithm

The algorithm that we propose is based on the notions defined in the previous paragraph : phases, synchronization and learning, and filterings. At the end of the algorithm a node will have received the information of all the nodes. Therefore we will need at least D phases to know all the information; in fact D phases are sufficient if the node has some global information like the diameter or the number of nodes. Otherwise a (D+1)-th phase is necessary in order the algorithm realize that it is in this situation. In fact our algorithm will have at most D+1 phases and is therefore optimal for the number of phases.

The algorithm is given for an arbitrary node P.

p : number of the current phase.

c : any channel incident to P.

Inf : The global information known by P (it is a set of the initial data of the nodes from which P has received information).

New : The new information obtained by P since the beginning of the current phase.

sent(c) : message sent on channel *c* at the current phase.

received(c) : message received through channel *c* at the current phase.

OPEN : set of the channels still open.

```
INIT      p ← 0
          Inf ← {initial data of the node}
          New ← Inf
          OPEN ← set of the channels incident to the node
          For every channel c  in OPEN  do  received(c) ← ø

PHASES    While OPEN ≠ ø do
          begin
            p ← p +1
            For every channel c  in OPEN  do
            begin
```

sent(c) ← *New* - *received(c)*

send <*sent(c)*> on *c*

end

New ← ø

For every channel *c* in *OPEN* do

begin

receive < *received(c)* > on *c*

If *received(c)* = *sent(c)* then *OPEN* = *OPEN* - {*c*}

New ← *New* ∪ (*received(c)* - *Inf*)

Call compute {compute depends on the application }

end

Inf ← *Inf* ∪*New*

end

TERM At the end of the algorithm the node knows all the useful information of the other nodes and by hypothesis the final result. *Inf* gives the set of data of all nodes and its cardinality gives the number of nodes.

3.2. Remark on the overtaking

We can suppose that the messages do not arrive in the order in which they were sent. We only need to add a phase counter to the message *sent(c)* indicating the number of the phase at which the message was sent. If the current phase of a node is *p* and if it receives a message <*received(c),p+1* >, then this message has to be put in a file and to be considered in the next phase (It follows from the definition that a node cannot send a message at phase *p+2* to a neighbor at phase *p*).

3.3. Proof

The intuitive idea of the proof consists in showing that during phase *p* a node P receives the information contained in the nodes at distance exactly *p* from itself. Therefore at the end of the phase *p* it knows the information of all the nodes at distance at most *p* from itself. Furthermore a channel will be closed at the end of a phase *p* if the two nodes joined by this channel have the same set of neighbours at distance at most *p-1* .

All the variables will concern node P

To simplify the proof let us introduce some notation.

$\Gamma(P)$ will denote the set of neighbours of P.

$\Gamma^p(P)$ will denote the set of nodes Q at distance at most p from P.

$N^p(P)$ will denote the set of nodes Q at distance exactly p from P. Note that $\Gamma^p(P)=U_{i\leq p}N^i(P)$.

We will denote by *Info* (A) the initial information contained in the set of nodes A.

Lemma. In the node P we have at the end of the phase *p* :

(i) *new* = *Info* ($N^p(P)$) or equivalently *Inf* = *Info* ($\Gamma^p(P)$)

(ii) the channel *c* joining P to Q is closed if and only if $\Gamma^{p-1}(P)=\Gamma^{p-1}(Q)$.

Proof. By induction on p

It is true for p=1; indeed P learns the information of all its neighbours (indeed at the beginning *OPEN* = the set of the channels incident to P). We do not close any channel joining two distinct nodes P and Q (if there was a loop in the network then we will close this loop which is in fact unuseful).

Suppose that the lemma is true for any $k \leq p$.

a) First we will prove (by induction) that the message *sent(c)* send by P to Q at the begining of phase *p+1* on the channel *c* joining P and Q satisfies the property *sent(c)* = *Info* ($N^p(P)$-$N^{p-1}(Q)$). That is clear for p=1, indeed *New* = *Info* ($\Gamma(P)$) and *sent(c)* = *Info* ($\Gamma(P)$-Q).

Recall that the message *sent(c)* send at the begining of the phase *p+1* equals *New-Received(c)* where these sets are obtained at the end of phase *p*. By induction hypothesis *New* = *Info* ($N^p(P)$) and *received(c)*= *Info* ($N^{p-1}(Q)$ -$N^{p-2}(P)$).

As $N^{p-2}(P)$ and $N^p(P)$ are disjoint, *sent(c)* = *Info* ($N^p(P)$ -$N^{p-1}(Q)$).

b) Now let us prove i).

At the end of the phase *p+1* the set *New* obtained in P satisfies:

New = $\cup_{c\in OPEN}$ *received(c)-Inf* . By induction hypothesis *Inf* = *Info* ($\Gamma^p(P)$).

If *c* is a channel joining P and Q, then *received(c)* = *Info* ($N^p(Q)$-$N^{p-1}(P)$) (as we have seen in a). As $\Gamma^p(P) \supseteq N^{p-1}(P)$, *New* = $\cup_{y\in O(P)}$ *Info* ($N^p(Q)$-$\Gamma^p(P)$) where O(P)={Q such that (PQ)∈ *OPEN*} (O(P) represent the set of neighbours which still communicate with P during the phase *p+1*). On the other hand :

$N^{p+1}(P) = \cup_{Q\in \Gamma(P)} (\Gamma^p(Q)-\Gamma^p(P))$. As $\Gamma^p(P)\supseteq \Gamma^{p-1}(Q)$ we have

$\Gamma^p(Q)-\Gamma^p(P) = N^p(Q)- \Gamma^p(P)$. To show that *New* = *Info* ($N^{p+1}(P)$), it remains to show that for any Q in $\Gamma(P)$-O(P), $\Gamma^p(Q)-\Gamma^p(P)$ is empty. The Q in $\Gamma(P)$-O(P) are exactly those for which the channel between P and Q has been closed at some phase i. By induction hypothesis that means that for some $i \leq p-1$ $\Gamma^i(P) = \Gamma^i(Q)$ and therefore $\Gamma^p(P) = \Gamma^p(Q)$.

c) Now let us prove ii)

Note that the property $\Gamma^p(P) = \Gamma^p(Q)$ is stable in that sense that if it is true for some p, then it is true for p+1. Suppose we close the channel *c* between P and Q at the phase p. Therefore *sent(c)=received(c)* or by the property seen in a) $N^p(P)-N^{p-1}(Q) = N^p(Q)-N^{p-1}(P)$.

Then $\Gamma^p(P)=\Gamma^{p-1}(Q) \cup N^p(P) \cup N^{p-1}(P) =\Gamma^{p-1}(Q) \cup (N^p(P)-N^{p-1}(Q)) \cup N^{p-1}(P)$

$=\Gamma^{p-1}(Q) \cup (N^p(Q)-N^{p-1}(P)) \cup N^{p-1}(P) = \Gamma^{p-1}(Q) \cup N^p(Q)= \Gamma^p(Q)$

Now suppose $\Gamma^{p-1}(Q)=\Gamma^{p-1}(P)$ and let us show that the channel *c* between P and Q will be closed at the end of phase p. As $\Gamma^{p-1}(Q)=\Gamma^{p-1}(P)$ then $N^p(Q)=N^p(P)$. Therefore *sent(c)* = *Info* ($N^p(P)-N^{p-1}(Q)$) = *Info* ($N^p(Q)-N^{p-1}(Q)$) = *Info* ($N^p(Q)$) (indeed $N^p(Q)$ and $N^{p-1}(Q)$ are disjoint) and *received(c)* = *Info* ($N^p(Q)-N^{p-1}(P)$) = *Info* ($N^p(P)-N^{p-1}(P)$) = *Info* ($N^p(P)$). Therefore *sent(c)=received(c)* .

Corollary . The algorithm terminates at phase D+1 and all the nodes know all the initial information of the others nodes.

Proof : Indeed for any P and Q $\Gamma^D(P) =\Gamma^D(Q)=V$ (recall V is the set of all nodes). Therefore at the end of the phase D+1 all the channels are closed and *OPEN* = Ø (ii) and *Inf* = *Info* ($\Gamma^D(P)$) = *Info* (V).

3.4. Complexity analysis

Lemma. The number of messages is at most 2(D+1)m.

Proof. Indeed during a phase a node sends a message on each open channel and receives a message from each open channel. Therefore there are at most 2m messages per phase (the total number of channels being m). Since the number of phases is less than or equal to D+1 for each node, the total number of messages is at most 2(D+1)m.

Remark. In fact we will see in the conclusion that D and m are related.

Lemma. The running time of the algorithm is bounded by $(2D+1)\tau$, where τ is the maximum transmission delay on a channel.

Proof. First note that all the nodes will be woken up within time τD (that is the maximal time necessary to wake up a node at distance D from the initiator). Since each phase has an execution time bounded by τ , the total running time of the phase part is at most $(D+1)\ \tau$. Therefore the total running time of the algorithm is at most $(2D+1)\ \tau$.

Remark: If we suppose the initial data to be of size logn, the size of a message is at most O(nlogn). The bit-complexity is O(nmlogn); indeed any information is transmitted on a channel at most twice (and at least once).

3.5. Applications

We have already seen how the algorithm can give a minimum routing table. Our algorithm has message complexity at most 2(D+1)m. Note that the best algorithms existing in the literature for the routing problem need a number of messages of order nm. As D is generally very small compare to m our algorithm is more efficient. At the end of the algorithm each node knows the identities of all the nodes and the data contained in them. Therefore it can easily compute the maximum of the identities or the sum of the data. We can also obtain the eccentricity of a node : it is equal to p_0 - 1, where p_0 is the first number of phase such that *New* becomes empty. To compute the diameter, it is enough to apply the algorithm a second time to compute the maximum of the eccentricities. Similarly to determine the centers of a graph (i.e. nodes with minimum eccentricity) it suffices to apply the algorithm a second time. Therfore the message-complexity to compute the centers is 2(D+1)m ; that improves the known bound and answers a problem of N.Santoro [S86]. More generally, we can solve any consensus problem.

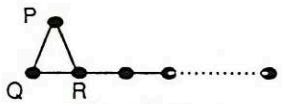
3.6. Remark

As claimed in 2.2 it is possible to refine the filtering used. Let c be the channel between two neighbours P and Q. From the point of view of P, if *sent(c)* is included in *received(c)*, then P will no longer send new information on channel *c* . Similarly if *received(c)* is included in *sent(c)* , then P will no longer learn new information from channel *c* . This leads us to define two sets of channels *IN* (set of channels open in emission) and *OUT* (set of channels open in reception) with the following handling : initially all the channels incident to P are in *IN* and *OUT*. At the phase *p*

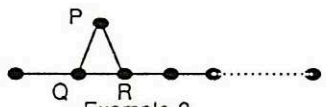
if *received(c)* $\supseteq$ *sent(c)* then P updates *OUT* to *OUT - c*

if *sent(c)* $\supseteq$ *received(c)* then P updates *IN* to *IN - c*.

The variant (given after) has the same complexities than the preceding one. However none of them is better than the other. Example 1 shows that the variant is better; indeed the channels from P to R and from Q to R are closed in the variant after phase 2 and therefore we earn a lot of empty messages (the channel between P and Q is completly closed in the two algorithms). In example 2, the channel from P to Q is closed in emission at the end of phase 2 in the variant; but the channel from Q to P will be used to send useless information until the end of the algorithm. In contrary in the original algorithm the channel is closed in both directions at the end of phase 3. Therefore less messages are sent on this channel in the original algorithm. Using this idea one can build networks for which the original algorithm is better than the variant.



Example 1



Example 2

INIT $p \leftarrow 0$

Inf $\leftarrow$ {initial data of the node}

New $\leftarrow$ *Inf*

IN $\leftarrow$ set of the channels incident to the node

OUT $\leftarrow$ set of the channels incident to the node

For every channel *c* in *IN* *received(c)* $\leftarrow \emptyset$

PHASES While *IN* $\neq \emptyset$ or *OUT* $\neq \emptyset$ do

begin

$p \leftarrow p+1$

For every channel *c* in *OUT* do

begin

if $c \in$ *IN* then *sent(c)* $\leftarrow$ *New - received(c)*

else *sent(c)* $\leftarrow$ *New*

send <*sent(c)*> on *c*

if *sent(c)* $= \emptyset$ then *OUT* $\leftarrow$ *OUT - c*

end

New $\leftarrow \emptyset$

For every channel *c* in *IN* do

begin

receive < *received(c)* > on *c*

if $c \in$ *OUT*

then if *received(c)* $\supseteq$ *sent(c)* then *OUT* $\leftarrow$ *OUT - c*

if *sent(c)* $\supseteq$ *received(c)* then *IN* $\leftarrow$ *IN - c*

else if *received(c)* $= \emptyset$ then *IN* $\leftarrow$ *IN - c*

New $\leftarrow$ *New* $\cup$ (*received(c)* - *Inf*)

Call compute {compute depends on the application }

end

Inf $\leftarrow$ *Inf* $\cup$ *New*

end

TERM At the end of the algorithm, *Inf* contains the information of all the nodes; P can compute the final result.

4. Conclusion

In this article we have given a general algorithm to solve consensus problems. As we have seen its message complexity is at most 2(D+1)m. Therefore it is important to minimize this product. However no node is privileged. Thus the number of channels incident to a node must be approximately constant. Furthermore for physical reasons, the number of channels incident to a node must be small. This leads us to construct networks with small diameter and given degree. Indeed a graph such that every vertex has degree Δ has $\Delta n/2$ edges.This problem is a well known one (see [BDQ86] for a survey). For example networks with diameter D, degree D and $n=(\Delta/2)^D$ vertices can be constructed. For such a network the

message complexity is $2(D+1)m=(D+1)\Delta m$ which is roughly $2D\ n^{1+1/D}$. This solves Lakshman and Agrawala Conjecture [LA86]. In their paper they solve the case when D=2. In fact with a good choice of parameters, we can reduce the message complexity to $O(n\log_2(n))$. In such a case our algorithm is very efficient. For further details showing the importance of network topology in distributed algorithms we refer the reader to [K87] or a forthcoming paper.

Acknowledgements: We thank very much C. Peyrat and N. Homobono for their smooth driving and careful reading of the paper.

REFERENCES

[A85] Awerbuch B.
"Complexity of Network synchronization", Journal of the ACM, 32, 4, 1985, 804-823.

[BDQ86] Bermond J-C., Delorme C., Quisquater J-J.
"Strategies for Interconnection Networks : Some Methods from Graph Theory", Journal of Parallel and Distributed Computing, 4 1986, 433-449.

[B87] Bougé L.
"Modularité et symétrie pour les Systèmes répartis", Thèse d'Etat, Paris, March 1987.

[HP85] Harel D., Pnueli A:
"On the development of reactive systems", in Logic and Models of Concurrent systems, NATO ASI Series F 13, Apt. Ed., 1985 477-498.

[G82] Gallagher R.G.
Distributed minimum hop algorithms.
MIT, Tech. Depart. LIDS-P-1175, 1982.

[K87] Konig J-C.
"Les réseaux d'interconnection et les algorithmes distribués", Thèse, Orsay, April 1987.

[LA86] Lakshman T.V., Agrawala A.K.
"Efficient decentralized Consensus Protocols", IEEE Trans. on Software Engineering, Vol SE12, 5, 1986, 600-607.

[R87] Raynal M.
"Systèmes Répartis et Réseaux, concepts outils et algorithmes", Eyrolles, 1987, 200 p.

[S86] Santoro N.
Problem 5, in Distributed Algorithms on Graphs, Proc 1st International Workshop on distributed algorithms, Carleton University Press, 1986,164.